Chapter 6
Toward an Anti-fragile e-Government System

Design is the process of defining a system's components, interfaces, data formats, data flows, and data storage solutions that together satisfy specified availability, performance, and scalability requirements. Chapter 4 introduced four design principles—modularity, weak links, redundancy, and diversity—and a single operational principle, fail fast, to achieve anti-fragility to a class of incidents. Chapter 5 showed how Netflix implemented the five principles to implement a media streaming system with anti-fragility to downtime. To investigate the generality of the five principles, the following two chapters investigate the design of systems to determine how they can be redesigned to achieve a degree of anti-fragility to downtime.

Here, we first study the Norwegian electronic government (e-government) system Altinn as it appeared in 2012 to better understand why it is advantageous to base the design of anti-fragile web-scale systems on fine-grained service-oriented architectures (SOAs) in public clouds with scalable and distributed data storage. This study is partly based on two analyses of Altinn commissioned by the Norwegian Ministry of Trade and Industry [57, 58]. Next, we consider the United Kingdom's e-government system to understand the need for user-focused and iterative development to support both rapid change and high availability. Finally, we discuss whether a nation should have a single e-government system running many services or multiple independent and diverse systems running a few services each.

6.1 The Norwegian e-Government System

The Norwegian government has made a large and sustained effort to develop world-leading e-government services for both citizens and companies [59]. Applications, invoicing, appointments, and various types of reports are all handled electronically. These digital services run on the Altinn platform. Sensitive personal information such as tax data are sent over the Internet to personal computing devices, including smartphones and tablets.

K.J. Hole, *Anti-fragile ICT Systems*, Simula SpringerBriefs on Computing 1,
DOI 10.1007/978-3-319-30070-2_6

Leading Norwegian politicians believe that most citizens prefer the new digital services to the old paper-based services [60]. The Web will be the primary communication channel between the Norwegian population and the public sector. While it will still be possible for private citizens to call or visit public sector offices, the government wants to minimize traditional person-to-person communications to free up resources needed to bring more and better health care services to Norway's aging population [59, 60]. Hence, high availability is increasingly important to Altinn as more services are added to the platform.

The 2012 version of the platform, denoted Altinn II, is depicted in Fig. 6.1a [57, 58]. A load balancer assigns requests to random servers that run user services. Note that the load balancer is a single point of failure. As illustrated in Fig. 6.1b, each user service is built on top of standardized components provided by the Altinn II platform. Multiple user services utilize the same component. Each component is assigned to a database. Altinn II allows government entities to develop and test their own user services. Scaling takes place by adding more servers and databases.

Despite the Altinn organization's best efforts, the platform's availability has been disappointingly low. The platform had to be taken offline for several days in 2011 and 2012 due to excessive network traffic when the Norwegian Tax Administration published the yearly tax statements. The damage to user trust was particularly noteworthy in 2012 because of the previous year's downtime and because many taxpayers were shown two individuals' names and national ID (IDentity) numbers when they tried to view their own tax statements. Due to some unknown failure, the names and ID numbers were cached and transmitted by the load balancer in Fig. 6.1a. While it can be argued that an ID number is not sensitive information, the national media reported extensively on this "crisis" and a later survey showed that the Altinn organization lost significant trust among its users. The incidents confirm the need to build and maintain trust, as discussed in Chap. 3.

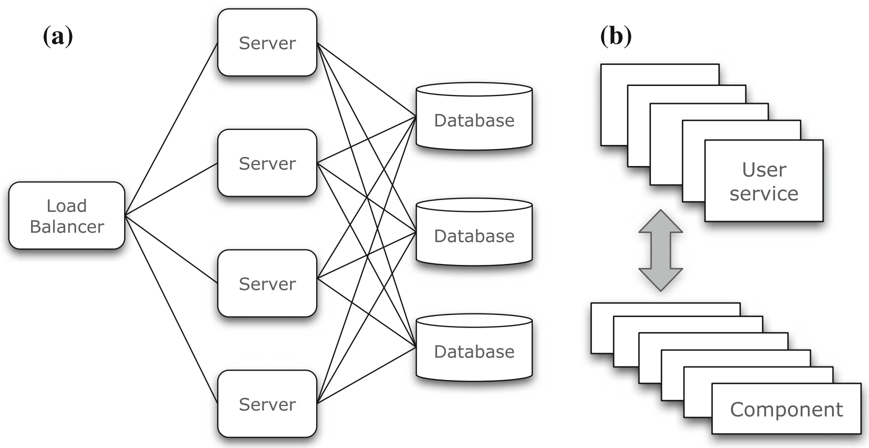

Fig. 6.1 **a** Altinn II architecture. **b** Layers of components and user services

6.2 Redesign Needed

The following analysis explains why the Altinn II platform must be redesigned to achieve anti-fragility to downtime. First, since each component in Fig. 6.1b is assigned a particular database, the speed at which a component can read or write data is limited by the capacity of a single database. As the amount of users and services increases, more processes will compete for access to each database because of the fixed component–database assignments [57, 58]. This database bottleneck makes it increasingly hard to scale the system by running more copies of the software on additional hardware. After some years, the system must be redesigned to handle more services and data.

Second, the Altinn II platform periodically runs batch jobs that lock up access to the databases. There is evidence that as the amount of data increases, the batch jobs will take longer to complete, further reducing the system's performance. Finally, Altinn II violates a fundamental tenet of SOA, namely, that user services should be independent of each other. Since dependencies exist between different user services, the failure of one service can negatively affect another. The dependencies also make it hard to upgrade certain user services without taking down the whole system.

In conclusion, while the design of Altinn II is supposedly based on SOA, the real design is database-centric with fragility to downtime due to the database bottleneck and the strong dependencies between software modules. The single load balancer is also problematic. A redesign based on the design principles of modularity, weak links, redundancy, and diversity introduced in Chap. 4 is needed to make the system robust to downtime. The redesign should be carried out by teams experienced in developing, operating, and maintaining web-scale systems, preferably using the development and operations (DevOps) methodology [37, 38]. The teams need to choose highly distributed and scalable data storage solutions to support additional user services with very high availability requirements.

6.3 Better Testing

It is much more demanding to test a platform that supports many services of national importance than to test a single enterprise application. The Altinn organization did not have an adequate test environment and test procedures in 2012 [57, 58]. The testing was inadequate in all phases of the development process and the ability to rectify discovered errors was limited. While Altinn was responsible for testing the components, the service owners were responsible for testing the services built on top of the components. The testing tools available to the service owners were not satisfactory. Because of insufficient testing, many bugs were not detected in the code before it went into production. In addition, due to budgetary constraints, many known defects in the production code were not rectified [57].

DevOps teams creating a new solution must improve testing during development. To learn from mistakes and achieve anti-fragility to downtime, the teams must also realize the fail fast operational principle introduced in Chap. 4.

6.4 Availability Requirements

A redesigned Altinn platform must satisfy well-defined availability, performance, and scalability requirements [57, 58]. Since this part of the book is concerned with anti-fragility to downtime, we only consider availability requirements. While the following requirements are obvious consequences of the problems with the Altinn II platform, they are valid for many types of platforms, including cloud computing platforms, where stakeholders run their own services:

1. A failure in a (user) service must not affect other services.
2. It must be possible to upgrade a service without system downtime.
3. It must be possible to launch a new service without system downtime.
4. Failure in a component must only affect services that use the component.
5. Good development and testing tools must be available to all developers.

6.5 Fine-Grained SOA in a Public Cloud

Achieving high availability is increasingly difficult the more complex a system becomes. Therefore, it is necessary to evolve a highly available web-scale solution, such as an e-government platform with important services, from a smaller, highly available system. Since no complex software system is perfect when first released, updates are needed to satisfy the availability requirements. It is both easier and faster to update the software when it is possible to change selected code modules without having to prepare a major new software release with all the testing it entails. Not all changes to a module lead to desired system behavior. Hence, weakly connected modules, that is, weak links, are essential to limit the negative impact of unintended behavior. Weak links let developers tinker with modules until the desired behavior is achieved. Tinkering makes it easier and faster to make the right changes and to avoid fragility to downtime at the same time.

The above discussion, as well as Netflix's experience discussed in Chap. 5, demonstrate that SOA with well-defined and self-contained services is an appropriate architecture for achieving anti-fragility to downtime. Furthermore, it makes sense to design microservices with limited functionality because it is hard to create default fallback responses for large services, each with much functionality [53, 54].

Since it is very expensive to acquire and maintain the computing hardware needed to support a web-scale solution, virtualization technology should be deployed to achieve reasonable costs [13]. Operating system-level virtualization supports a highly scalable system of many independent computing devices, making it possible to allocate and use idle computing resources more efficiently than in a traditional datacenter without virtualization technology. Rather than building and maintaining a private cloud, a government should seriously consider using a public cloud because its total cost is divided among many entities. We conclude the following:

- To achieve anti-fragility to downtime, governments should base e-government platforms and services on a fine-grained SOA in public clouds with highly redundant and scalable data storage.

6.6 User-Focused and Iterative Development

The external Altinn evaluations [57, 58] indicate a need for more user-focused and iterative development to adapt services to users' needs and make the services easier to use [37, 38]. The ability to quickly modify software is also important to mitigate problems and achieve anti-fragility to downtime. In the United Kingdom, there is a unit within the Cabinet Office, called the Government Digital Service (GDS), tasked with transforming government digital services according to users' needs (http://gds.blog.gov.uk). The GDS works with government departments to develop user services, promotes open source development philosophies, and ensures that services are built on open standards and application programming interfaces. The reader can find more information about GDS's development methodology in the Digital by Default Service Standard and its accompanying manual (http://gov.uk/service-manual).

The GDS utilizes the DevOps methodology to break down the traditional silos of development, quality assurance, and operations. The goal is to foster an attitude of shared ownership and collaboration, resulting in common working practices in designing and operating a software solution (http://infoq.com/news/2015/03/gds-uk-gov-devops). GDS has successfully moved the Web presence of all UK government departments to gov.uk [61]. This Web platform publishes government information and provides access to online services. To build e-government services that citizens will use, the GDS has found that developers first need to thoroughly understand the users' needs. Rather than make assumptions, developers must analyze real data from similar services and interview future users to determine their needs. To maintain usability, developers need to revisit services and make alterations as users' needs change over time.

Any service should be designed around the identified users' needs. According to the GDS, developers should start small and iterate often. Frequent iterations reduce the probability of big failures and turn small failures into lessons. It is essential to release prototype solutions early, test them with real users, and move from alpha to beta releases while adding features and refinements based on user feedback.

Viewing e-government infrastructures as complex adaptive systems partly explains the GDS's success with the DevOps methodology. Since it is very hard to predict the long-term global behavior of complex systems, iterative and test-driven approaches are needed to ensure the sufficient availability, scalability, and performance of new services. Experience with Altinn shows that insufficient testing leads to undetected errors, causing trust-reducing incidents [57]. Even worse, shortcomings of the underlying platform architecture go undetected, making it increasingly difficult and costly to satisfy the design requirements as the numbers of users and services grow.

6.7 Single Versus Multiple Systems

Should a nation have a single e-government platform running many services or multiple diverse and independent platforms running a few services each? The following answer builds on arguments first presented in an earlier paper [34]. In all the cases discussed, the services access sensitive personal information, including financial and medical data.

6.7.1 Systems with Strongly Connected Modules

We first consider e-government systems with strongly connected modules. Each system runs in a traditional datacenter without cloud technologies. We compare a scenario in which a nation employs a single system providing many services with a scenario in which the same nation uses multiple diverse systems providing a few services each. As long as no swan events cause prolonged downtime, the single system exploits economies of scale. Additionally, users enjoy a high degree of usability because the single system lets them authenticate with numerous services using the same authentication technique. Finally, a common user interface design for all services further enhances usability and helps attract many users.

Deploying diverse e-government systems increases the overall burden of system management. The user experience suffers because users must relate to multiple authentication techniques and user interface designs. Consequently, the number of citizens using the online services can decrease. Therefore, a single system is the preferred scenario, barring any black or gray swans.

The situation changes radically when a swan occurs that leads to prolonged downtime. Suddenly, the government is in trouble with millions of citizens, who all want to know what happened and what the government is going to do about the intolerable situation. Because no alternative to the single system exists, it can take a long time before financial and medical information becomes available again. The delay can cause intolerable problems for all users dependent on the information. Therefore, in a swan-prone world, multiple diverse e-government systems are significantly less risky to major stakeholders than a single centralized system, as long as we compare e-government systems with strongly connected modules that run in traditional datacenters.

The choice between building a single or multiple e-government systems with strongly connected modules is essentially the choice between accepting rare, catastrophic events and more frequent, less serious incidents. A nation should, therefore, employ a single strongly connected system only after a thorough and comprehensive risk analysis concludes that all major stakeholders can tolerate swans. If a single system is the solution, the system owner can improve resistance to swans by removing single points of failure and decreasing susceptibility to cascade failures. However, it will be very expensive to obtain the system redundancy and diversity needed to significantly reduce the probability of swans.

While the risk of swans is mitigated by deploying multiple independent and diverse e-government systems running a few services each, it may also be necessary to have two systems delivering the same critically important service. The experience of the Norwegian Food Safety Authority demonstrates the advantage of an alternative solution when a system goes down. The Authority provides a service on the Altinn II platform to allow Norwegian fish exporters obtain export licenses. The Authority produces export licenses for fish worth about 60 billion Norwegian kroner (NOK) in a year. When Altinn went down for several days in 2012, the Authority used an alternative service to produce export licenses, thus protecting the Norwegian fishing industry from large financial losses.

6.7.2 Cloud-Based Systems of Weakly Connected Modules

The main advantage of the cloud is that an application owner can achieve the redundancy and diversity of multiple independent implementations in traditional datacenters without actually having to develop and maintain multiple diverse applications. Large public clouds, such as Google App Engine, Amazon Web Services, and Microsoft Azure, are highly geographically distributed infrastructures with regions on different continents, each with multiple zones (datacenters). As demonstrated by Netflix, a cloud-native solution that takes full advantage of a cloud's services can create a single application with very high uptime. Hence, there is much less need to implement multiple independent and diverse versions of an application when it is possible to make a cloud-native application.

While a single cloud platform provides the needed uptime for most services, governmental e-voting services that allow citizens to vote over the Internet during general elections may be an exception to the rule of using only one cloud platform. Since there are indications that citizens are particularly sensitive to failures in voting systems, a government should have an independent voting alternative in case a cloud-based e-voting service has a major outage. In Norway, citizens in certain municipalities have been allowed to vote over the Internet for multiple weeks during elections [7, 43]. The long voting period was selected to reduce the impact of shorter outages because citizens could easily vote later. Furthermore, any citizen could cast a paper vote on the traditional election day, invalidating any earlier cast e-vote. Hence, e-voting did not replace paper-based voting; it only provides an additional means of voting.

To conclude, it is necessary to have multiple and diverse e-government platforms running relatively few services each when each platform solution is strongly connected and run in a traditional datacenter. When a cloud-native solution provides the e-government services, one solution is enough, unless we are talking about very critical e-government services such as e-voting for national elections. In this case, the most critical services need to have independent backup solutions [43, 62]. The reader should keep in mind that no solution can scale forever; scaling will eventually introduce fragility to downtime because the inevitable increase in complexity will introduce unforeseen and fragilizing dependencies [2].

6.8 Discussion and Summary

An information and communications technology (ICT) system's mean time to repair (MTTR) is the average time from when a failure occurs until it is repaired and the mean time between failures (MTBF) is the average time between two consecutive failures. While operators of traditional monolithic ICT systems typically try to minimize the MTTR or maximize the MTBF, this is both difficult and costly because the strong dependencies between system modules facilitate local failure propagation leading to expensive systemic failures. There is a need for a better way to build and operate complex adaptive ICT systems with a degree of anti-fragility to downtime.

The increasing popularity of cloud computing and the DevOps methodology facilitate the realization of SOA with microservices that model software applications as sets of independently deployable and scalable services with well-defined interfaces [53, 54]. Circuit breakers remove much of the problem of cascading failures and the use of microservices with limited functionality makes it possible to ensure the graceful degradation of an application's functionality. The limited functionality of each service facilitates the development of automated fallback responses in the case of local failures. When a local failure affects a service, other services depending on this malfunctioning service receive a standardized response. This architectural style also supports the development and management of services by multiple teams using different programming languages; continuous deployment, enabling rapid innovation; and highly redundant and scalable data storage, making data loss extremely unlikely.

While traditional monolithic solutions struggle to achieve high availability, the success of Netflix and other organizations such as Nike and the British newspaper *The Guardian* show that SOA with microservices is well suited to ICT infrastructures requiring high availability. However, it is too early to conclude that this architectural style is the future of huge enterprise and governmental solutions because serious weaknesses may first emerge only after solutions have been in production for years.

For now, we conclude that cloud-based solutions with SOA and microservices can achieve anti-fragility to downtime. At the same time, it seems more difficult and much more expensive to build monolithic applications outside the cloud with anti-fragility to downtime.